BASTION: Budget-Aware Speculative Decoding with Tree-structured Block Diffusion Drafting

Soowon Oh [*1,2] **Nam Cao** [*1] **Yujin Kim** [1] **Hojung Jung** [1] **Huzama Ahmad** [1] **Sangmin Bae** [†1] **Se-Young Yun** [†1]

Abstract

Block-diffusion drafters have recently emerged as a powerful alternative for speculative decoding by predicting multiple future-token distributions in a single parallel step. However, since these parallel predictions are sampled from position-wise marginals rather than fully conditioned sequences, committing to a single greedy path often fails to capture the target model's preferred trajectory. To address this, we propose **BASTION**, a **b**udget-**a**ware **s**peculative decoding framework with **t**ree-based diffus**ion** drafting. Unlike existing methods that rely on static tree topologies, BASTION dynamically constructs query-dependent trees by balancing draft quality against hardware constraints. Our framework integrates three synergistic components: (1) an *acceptance surrogate* that estimates expected accepted length via path confidence, (2) an *online latency estimator* that calibrates a hardware-aware roofline model, and (3) an *adaptive best-first expansion* that grows the tree until marginal gains no longer justify incremental verification costs. BASTION is training-free, preserves the target model's distribution, and requires no per-setting tuning. Across diverse benchmarks and GPU architectures, BASTION achieves up to a $6.61\times$ speedup over standard autoregressive decoding, outperforming state-of-the-art block-diffusion baselines by 39%.

1. Introduction

Recent large language models (LLMs) demonstrate impressive reasoning capabilities (Grattafiori et al., 2024; Comanici et al., 2025; Yang et al., 2025; Agarwal et al., 2025; DeepSeek-AI, 2026), but their serving costs remain high because autoregressive (AR) decoding requires a separate target-model forward pass for every generated token. *Speculative decoding* mitigates this bottleneck by using a fast drafter—either a smaller auxiliary model (Chen et al., 2023; Leviathan et al., 2023; Kim et al., 2023; Miao et al., 2024) or a self-drafting mechanism (Bae et al., 2023; Elhoushi et al., 2024; Cai et al., 2024; Li et al., 2024a; Zhang et al., 2024a)—to propose multiple candidate tokens, which the target model verifies in parallel with an exact acceptance rule (Chen et al., 2023; Leviathan et al., 2023) or a relaxed variant (Cai et al., 2024; Jang et al., 2024; Bachmann et al., 2025). Under exact acceptance, this preserves the target distribution while shifting generation from token-by-token decoding to batched verification, with speedup governed by the acceptance rate relative to drafting and verification overhead.

Most prior methods optimize this trade-off within an autoregressive drafting paradigm (Leviathan et al., 2023; Sadhukhan et al., 2024; Elhoushi et al., 2024; Li et al., 2025b), designing lightweight yet context-aware drafters. State-of-the-art methods such as EAGLE-3 (Li et al., 2025b) deliver practical speedups in the 2–3$\times$ range. However, because their drafting phase remains sequential, longer candidate blocks require repeated draft-model steps conditioned on previous draft tokens, making the draft stage increasingly costly and underutilizing available parallel compute.

To break this sequential dependency, recent works explore multi-token prediction (Gloeckle et al., 2024; Samragh et al., 2025; Liu et al., 2025a; Chen et al., 2026; Kirchenbauer et al., 2026), where multiple future tokens are proposed in a single drafting step. Block diffusion language models (Nie et al., 2025; Arriola et al., 2025; Wu et al., 2025b;a) provide a natural instantiation: a diffusion drafter denoises a fixed block of mask tokens simultaneously, producing candidate distributions over future positions without left-to-right generation. This parallel structure is well suited to speculative decoding, as shown by TiDAR (Liu et al., 2025a) and DFlash (Chen et al., 2026), which report up to 2.5$\times$ speedups over AR drafting baselines.

Despite these latency benefits, parallel prediction introduces a key challenge: future positions are drafted before

[*]Equal contribution [1]KAIST AI, Seoul, Republic of Korea [2]Samsung Advanced Institute of Technology, Suwon, Republic of Korea. Correspondence to: Sangmin Bae <bsmn0223@kaist.ac.kr>, Se-Young Yun <yunseyoung@kaist.ac.kr>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

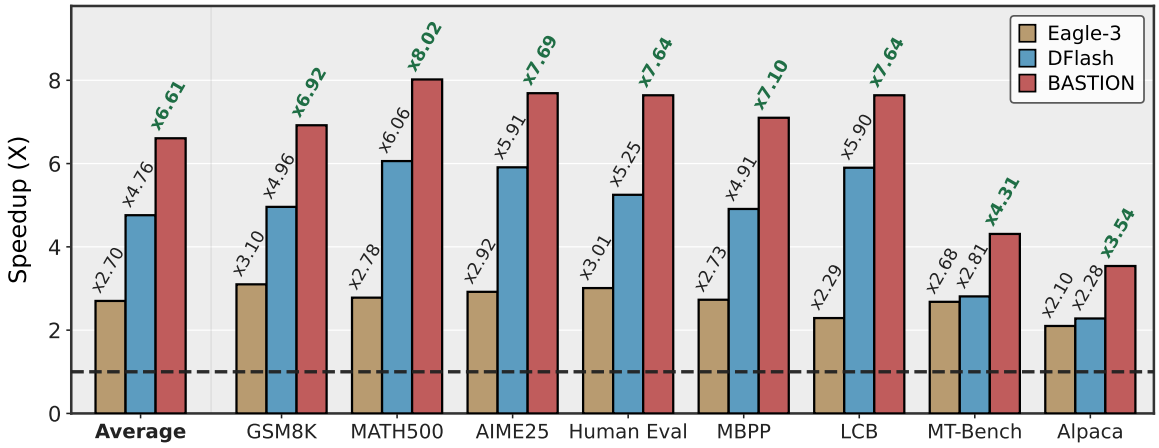


Figure 1. **BASTION achieves a 6.61× average end-to-end speedup on Qwen3-8B.** BASTION consistently outperforms speculative decoding baselines (EAGLE-3 (Li et al., 2025b) and DFlash (Chen et al., 2026)) across eight diverse benchmarks (three math, three code, and two chat datasets). The baseline performance (1×) represents standard autoregressive decoding. Results are evaluated for a single sample using greedy decoding (i.e., temperature of 0) on an A100 GPU.

preceding tokens are finalized, so the outputs correspond to position-wise marginals rather than fully conditioned AR continuations. Consequently, greedy top-1 sampling can yield incoherent sequences under the target model's distribution (Gu et al., 2017; Li et al., 2022; Han et al., 2023; Wu et al., 2023). Yet the target-preferred trajectory often remains within the drafter's top-K candidates. Motivated by this, we propose ***tree*-structured block-diffusion drafting**: instead of committing to one greedy path, we construct a prefix tree from the parallel candidate distributions, allowing the target model to verify multiple plausible trajectories simultaneously. This inherits the core advantages of tree-based speculative decoding (Miao et al., 2024; Cai et al., 2024; Li et al., 2024b), improving verifier utilization and expected acceptance length.

However, the efficiency of tree-based drafting depends critically on its topology. Prior methods typically rely on heuristically predefined, static tree shapes that remain fixed across inputs and hardware constraints (Cai et al., 2024; Li et al., 2024a; Park et al., 2025). To address this, we propose **BASTION**, a **b**udget-**a**ware **s**peculative decoding method with **t**ree-based diffus**ion** drafting. Unlike static approaches, BASTION dynamically constructs a query-dependent tree topology to maximize expected decoding speedup. It combines three components: **(1)** an *acceptance surrogate* that estimates expected accepted length from path confidence scores under the drafter's marginal distributions; **(2)** an *online latency estimator* that predicts verification cost with a hardware-aware roofline model calibrated by observed runtimes; and **(3)** an *adaptive best-first expansion* mechanism that grows the tree until the marginal acceptance gain no longer outweighs the added verification cost.

By balancing draft quality against hardware capacity, BASTION delivers substantial empirical speedups. In Figure 1, evaluations across math, code generation, and chat benchmarks show average speedups of 6.61× over standard decoding, 2.45× over EAGLE-3 (Li et al., 2025b), and 1.39× over DFlash (Chen et al., 2026). These gains are consistent across multiple model and GPU architectures, highlighting the practical deployment viability of dynamic tree-structured drafting.

Contributions. In summary, our key contributions in this paper are as follows:

- **Tree-based block diffusion drafting:** We introduce a best-first prefix-tree construction algorithm that uses drafter-logit path scores to build high-quality prefix-closed candidate trees from block-diffusion logits.
- **Hardware-aware online budget controller:** To overcome the inefficiencies of fixed-size trees, we introduce a budget controller that dynamically determines the optimal tree size at each decoding step by balancing the expected acceptance gain against hardware-calibrated verification latency.
- **Extensive empirical validation:** We validate BASTION across target models, benchmarks, GPU architectures, and decoding temperatures, showing up to 6.61× speedup over autoregressive decoding and a 38.87% improvement over single-path block-diffusion drafting.

2. BASTION: Budget-Aware Tree Construction for Diffusion Drafting

In this section, we introduce our method **BASTION**, a **b**udget-**a**ware **s**peculative decoding framework with **t**ree-based diffus**ion** drafting. This section formalizes three com-

ponents: (i) a problem formulation that exploits the parallel nature of block-diffusion drafting (§2.1), (ii) a path-confidence surrogate for acceptance estimation (§2.2,§2.3), and (iii) an online controller for budget optimization (§2.4).

2.1. Problem Formulation: Tree-based Block-Diffusion Drafting

A block-diffusion drafter produces position-wise marginal distributions $\{q_k(\cdot)\}_{k=1}^{\gamma}$ in a single parallel forward pass at constant cost $T_{\text{draft}} = t_{\text{parallel}}$, whereas an *autoregressive* (AR) drafter incurs $T_{\text{draft}} = \gamma \cdot t_{\text{step}}$. Two consequences shape our design:

(P1) **Topology decoupling.** T_{draft} is invariant to the candidate-set structure—chains, wide trees, and deep trees incur identical drafting cost.

(P2) **Marginal independence.** Each q_k is conditioned on the validated context alone, not on drafted tokens at positions $\{1, \ldots, k-1\}$, so candidates across positions can be combined freely.

These properties motivate organizing the candidates as a *prefix tree* $\mathcal{T}$: the root r is the validated bonus token, and each non-root node i at depth $d(i) \in \{1, \ldots, \gamma\}$ carries a candidate token x_i whose root-to-node path $\text{path}(i) = (x_{i_1}, \ldots, x_{i_{d(i)}})$ receives the *path score*

$$\rho(i) \;=\; \prod_{k=1}^{d(i)} q_k(x_{i_k}), \qquad \rho(r) = 1. \tag{1}$$

The target model verifies all nodes in one parallel forward pass and commits the longest accepted root-to-node path; counting the always-committed root, let $A(\mathcal{T})$ be the resulting committed length and define the *verification budget* $N = |\mathcal{T}|$. The end-to-end speedup over AR decoding is then

$$\text{Speedup}(\mathcal{T}) \;=\; \frac{\mathbb{E}[A(\mathcal{T})] \cdot L_{\text{AR}}}{T_{\text{draft}} + T_{\text{aux}}(\mathcal{T}) + T_{\text{verify}}(\mathcal{T})}, \tag{2}$$

where L_{AR} is one AR target step, $T_{\text{aux}}(\mathcal{T})$ covers tree construction and scheduling, and $T_{\text{verify}}(\mathcal{T})$ is the parallel verification cost. Our goal is to choose $\mathcal{T}$—both its shape and size N—to maximize (2) from the drafter's logits alone.

2.2. Expected Committed Length Surrogate

Maximizing (2) before verification is intractable: the realized committed length $A(\mathcal{T})$ depends on the target's outputs, which are unavailable until verification completes. We therefore approximate $\mathbb{E}[A(\mathcal{T})]$ via a *drafter-side surrogate* obtained from a self-verification thought experiment.

Self-verification model. Replace the target with the drafter itself as the verifier: the drafter draws a single sample $X = (X_1, \ldots, X_\gamma)$ with $X_k \sim q_k(\cdot)$ independently across positions, and a node $i \in \mathcal{T}$ is committed iff $X_{1:d(i)} = \text{path}(i)$. By property (P2), this product distribution is *precisely* the drafter's joint distribution over γ-length continuations, so this model is the natural drafter analogue of target verification—not an approximation of the drafter, only of the target. Let $A_{\text{self}}(\mathcal{T}; X)$ denote the resulting committed length (including the always-committed root).

Lemma 2.1 (Committed length equals covered count)**.** *Under the self-verification model,*

$$A_{\text{self}}(\mathcal{T}; X) \;=\; \#\{\, i \in \mathcal{T} \;:\; X_{1:d(i)} = \text{path}(i) \,\}.$$

The set of committed nodes forms a contiguous root-to-node chain in $\mathcal{T}$.

The proof (Appendix C.1) follows from two structural facts: prefix-closure of $\mathcal{T}$ forces the covered set to be ancestor-closed, and the tree structure permits at most one covered node per depth; hence the covered set is a single chain whose length equals its cardinality.

Proposition 2.2 (Path-Sum Surrogate)**.** *Taking expectation over the drafter's sample X,*

$$\widehat{A}(\mathcal{T}) \;\triangleq\; \mathbb{E}_X[A_{\text{self}}(\mathcal{T}; X)] \;=\; \sum_{i \in \mathcal{T}} \rho(i). \tag{3}$$

The proof (Appendix C.1) applies linearity of expectation to Lemma 2.1, using $\mathbb{P}(X_{1:d(i)} = \text{path}(i)) = \rho(i)$ by position-wise independence and the convention $\rho(r) = 1$ for the trivially covered root.

Why $\widehat{A}$ proxies the target-side expectation. (3) is *exact* only under self-verification, where the drafter acts as both proposer and verifier. Its use as a proxy for the target-verified $\mathbb{E}[A(\mathcal{T})]$ instead relies on drafter–target alignment, the foundational premise of speculative decoding: when the drafter's high-probability continuations coincide with the target's, the drafter-mass covered by $\mathcal{T}$ closely tracks the realized accepted length. Appendix H.1 (Figure 6) empirically validates this proxy with Pearson correlation ≥ 0.79 across all target/draft pairs.

2.3. Optimal Tree Construction via Best-First Expansion

With the surrogate in hand, the fixed-budget tree-construction problem becomes: *given N, find the prefix-closed tree $\mathcal{T} \subseteq \mathcal{V}$ of size N that maximizes $\widehat{A}(\mathcal{T})$,* where $\mathcal{V}$ denotes the candidate lattice induced by retaining the top-K tokens at each position. Beam-search-style methods, which

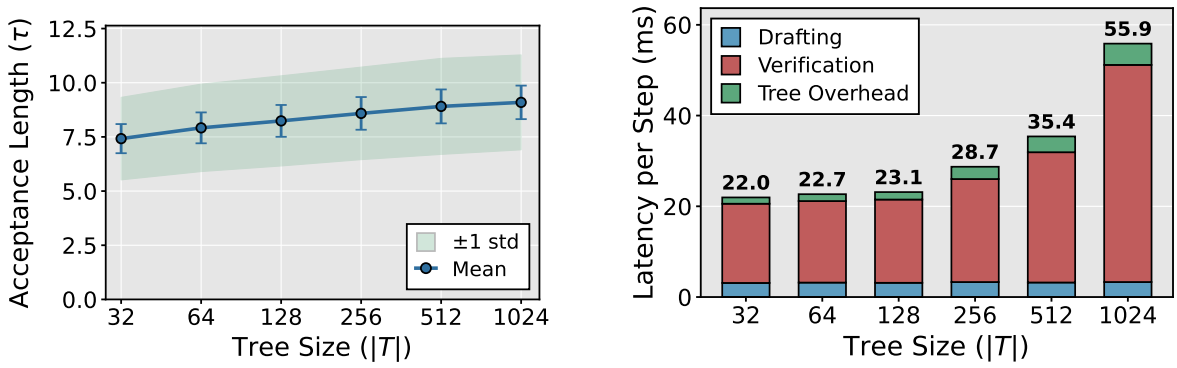
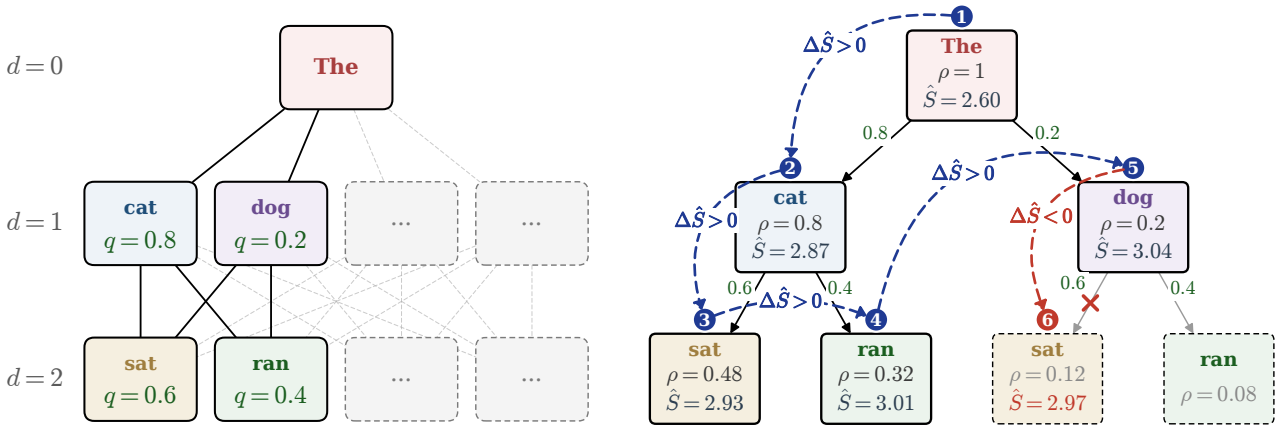


Figure 2. **Acceptance–latency trade-off across tree sizes.** *Left:* acceptance length τ grows with tree size $|\mathcal{T}|$ but saturates beyond a few hundred nodes, reflecting diminishing marginal gains. *Right:* per-step latency breakdown—drafting cost is constant, while T_{aux} and T_{verify} grow with $|\mathcal{T}|$, with T_{verify} dominating at large budgets (22.0 ms at $|\mathcal{T}|$=32 rising to 55.9 ms at $|\mathcal{T}|$=1024).



(a) Top-K logits per position

(b) Adaptive tree construction

Figure 3. **Adaptive tree construction from block-diffusion logits. (a)** The drafter provides top-K candidates for multiple future positions in one forward pass, inducing an implicit lattice of candidate prefixes. **(b)** Best-first expansion adds nodes in descending path probability $\rho(i)$ and evaluates the estimated speedup $\widehat{S}_t(N)$ after each intermediate budget. The controller returns the tree with the largest estimated speedup instead of using a fixed depth or width.

fix a (width $\times$ depth) allocation, are mismatched here because they ignore the global ordering of path confidences across depths.

A key structural property is *path monotonicity*: for any non-root i with parent $\pi(i)$,

$$\rho(i) \;=\; \rho(\pi(i)) \cdot q_{d(i)}(x_i) \;\leq\; \rho(\pi(i)), \tag{4}$$

because $q_{d(i)}(x_i) \in [0, 1]$. This monotonicity, combined with prefix-closure, makes a greedy procedure provably optimal.

Proposition 2.3 (Optimality of Best-First Expansion)**.** *Iteratively adding the valid node with the largest ρ yields nested trees $\mathcal{T}_1 \subset \mathcal{T}_2 \subset \cdots \subset \mathcal{T}_{N_{\max}}$ where each $\mathcal{T}_N$ maximizes $\widehat{A}(\mathcal{T})$ among all size-N prefix-closed trees drawn from $\mathcal{V}$.*

The proof (Appendix C.2) demonstrates that best-first enumerates nodes in a globally non-increasing order of ρ. Consequently, the marginal gain $\Delta\widehat{A}(N) := \widehat{A}(\mathcal{T}_{N+1}) - \widehat{A}(\mathcal{T}_N) = \rho(i_{N+1})$ is non-increasing:

$$\Delta\widehat{A}(N+1) \;\leq\; \Delta\widehat{A}(N) \quad \text{for all } N, \tag{5}$$

making $\widehat{A}$ *concave* in the budget N. As shown in Figure 2 (left), this concavity yields diminishing returns, directly motivating the need for an adaptive budget strategy.

2.4. Online Controller for Budget Optimization

For fixed-budget construction to maximize $\widehat{A}(\mathcal{T})$ with a given N, no single N is universally optimal: the marginal acceptance gain depends on the local draft distribution, while

Table 1. **Comparison of speedup and AAL (τ) on NVIDIA A100 gpu.** The experiments are conducted on Qwen3-4B and Qwen3-8B at Temperature $\in \{0, 1\}$. For the baselines, the tree budget size of EAGLE-3 is set to 60, and the block size of DFlash is set to 16.

Model	Method	GSM8K		MATH500		AIME25		HumanEval		MBPP		LCB		MT-Bench		Alpaca		Average	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Greedy Decoding (Temperature = 0)																			
Q3-4B	EAGLE-3	3.11×	2.77	2.84×	2.53	2.92×	2.52	2.93×	2.57	2.77×	2.48	2.66×	2.10	2.69×	2.48	2.45×	2.15	2.80×	2.45
	DFlash	5.23×	6.50	6.22×	8.12	6.16×	7.48	5.34×	6.66	5.09×	6.31	5.71×	7.23	2.74×	4.63	2.12×	2.90	4.83×	6.23
	BASTION	**7.44×**	**9.07**	**8.57×**	**10.60**	**8.23×**	**9.70**	**7.91×**	**9.38**	**7.41×**	**8.99**	**7.81×**	**9.48**	**4.53×**	**6.58**	**3.53×**	**4.64**	**6.93×**	**8.56**
Q3-8B	EAGLE-3	3.10×	2.74	2.78×	2.51	2.92×	2.45	3.01×	2.69	2.73×	2.24	2.29×	1.91	2.68×	2.24	2.10×	1.77	2.70×	2.32
	DFlash	4.96×	6.51	6.06×	8.25	5.91×	7.20	5.25×	6.64	4.91×	6.04	5.90×	7.64	2.81×	4.46	2.28×	3.15	4.76×	6.24
	BASTION	**6.92×**	**8.85**	**8.02×**	**10.53**	**7.69×**	**9.46**	**7.64×**	**9.44**	**7.10×**	**8.65**	**7.64×**	**9.91**	**4.31×**	**6.28**	**3.54×**	**4.68**	**6.61×**	**8.48**
Stochastic Decoding (Temperature = 1)																			
Q3-4B	EAGLE-3	3.03×	2.75	2.69×	2.44	2.61×	2.21	2.61×	2.44	2.70×	2.41	2.38×	2.08	2.59×	2.37	2.41×	2.16	2.63×	2.36
	DFlash	4.71×	5.89	5.18×	6.85	3.91×	5.08	4.60×	6.04	4.73×	5.88	5.10×	6.90	2.65×	4.21	2.10×	2.76	4.12×	5.45
	BASTION	**7.38×**	**8.91**	**8.44×**	**10.59**	**7.49×**	**9.16**	**7.09×**	**9.09**	**7.07×**	**8.77**	**7.69×**	**9.83**	**4.55×**	**6.57**	**3.56×**	**4.70**	**6.66×**	**8.45**
Q3-8B	EAGLE-3	2.86×	2.52	2.76×	2.35	2.49×	2.12	2.63×	2.49	2.52×	2.23	2.28×	1.84	2.45×	2.04	2.02×	1.65	2.50×	2.16
	DFlash	4.56×	5.80	4.98×	6.54	3.70×	4.72	4.44×	5.75	4.13×	5.44	5.21×	6.91	2.60×	4.00	2.09×	2.90	3.96×	5.26
	BASTION	**7.05×**	**8.70**	**8.51×**	**10.41**	**7.35×**	**9.21**	**6.98×**	**9.12**	**6.75×**	**8.78**	**7.81×**	**9.82**	**4.32×**	**6.32**	**3.46×**	**4.64**	**6.53×**	**8.38**

the verification cost depends on context length, model architecture, and hardware. As illustrated in Figure 2 (right), verification costs T_{verify} scale with $N = |\mathcal{T}|$ and eventually dominate the cycle. To formalize this trade-off, we maximize the *estimated speedup at budget* N:

$$S(N) \;=\; \frac{\widehat{A}(\mathcal{T}_N) \cdot L_{\text{AR}}}{C(N)} \tag{6}$$

$$\begin{aligned} C(N) \;&\triangleq\; T_{\text{draft}} + T_{\text{aux}}(N) + T_{\text{verify}}(N) \\ &\approx T_{\text{draft}} + T_{\text{aux}} + T_{\text{verify}}(N) \end{aligned} \tag{7}$$

Based on empirical observations from Figure 2 (right), $T_{\text{aux}}(N)$ is negligible compared to the verification overhead, allowing us to approximate it as a constant T_{aux}. Crucially, we predict $T_{\text{verify}}(N)$ using an analytical hardware-aware cost model based on roofline analysis (detailed breakdown in Appendix F). This analytical modeling reveals that $T_{\text{verify}}(N)$, and thus the overall cycle cost $C(N)$, is convex with respect to N. Additionally, since practical latency may deviate from analytical estimation due to factors such as kernel fusion, cache effects, scheduling overhead, and implementation details of tree attention, we statically apply a linear calibration to the cost model for more precise latency prediction.

Proposition 2.4 (Unimodality of Estimated Speedup)**.** *Suppose $\widehat{A}(N)$ is concave and non-decreasing, and $C(N)$ is convex and strictly positive. Then $S(N)$ is unimodal on $\{1, \ldots, N_{\max}\}$. In particular, the smallest N for which $S(N{+}1) < S(N)$ is a global maximizer.*

The proof is provided in Appendix C.3. Propositions 2.3 and 2.4 combine into a single online procedure as illustrated in Figure 3. Best-first expansion produces a nested chain $\mathcal{T}_1 \subset \mathcal{T}_2 \subset \cdots$, with each step adding one node i_{N+1} and updating the surrogate incrementally as $\widehat{A}(\mathcal{T}_{N+1}) = \widehat{A}(\mathcal{T}_N) + \rho(i_{N+1})$. We pair this with a hardware-calibrated estimate $\widehat{C}(N)$ of cycle latency and track $\widehat{S}(N) = \widehat{A}(\mathcal{T}_N)\, L_{\text{AR}}/\widehat{C}(N)$. By Proposition 2.4, the controller halts at the first N where $\widehat{S}(N{+}1) < \widehat{S}(N)$ and returns $\mathcal{T}_N$. The per-step overhead is dominated by best-first heap operations and is negligible compared to the target forward pass. Pseudocode is given in Algorithm 1.

3. Experiments

3.1. Experimental Settings

We evaluate BASTION using Qwen3 (4B/8B) and Llama-3.1-8B-Instruct paired with DFlash block-diffusion drafters. We compare our method against DFlash and EAGLE-3 across four primary domains: mathematical reasoning, code generation, instruction following, and long-context understanding. Detailed experimental settings, including comprehensive dataset lists, metrics, and hardware configurations, are deferred to Appendix G.

3.2. Main Results

End-to-end speedup. Table 1 reports comparative results across two Qwen3 target models, two decoding temperatures, and eight benchmarks on NVIDIA A100. BASTION improves speedup over both DFlash and EAGLE-3 in all settings, with the largest gains on math and code benchmarks. Compared with DFlash, the improvement comes from using the drafter's full position-wise distribution to construct a tree rather than verifying only a single top-1 continuation. Compared with EAGLE-3, the gain reflects the lower draft latency of block-diffusion drafting together with adaptive tree-budget selection.

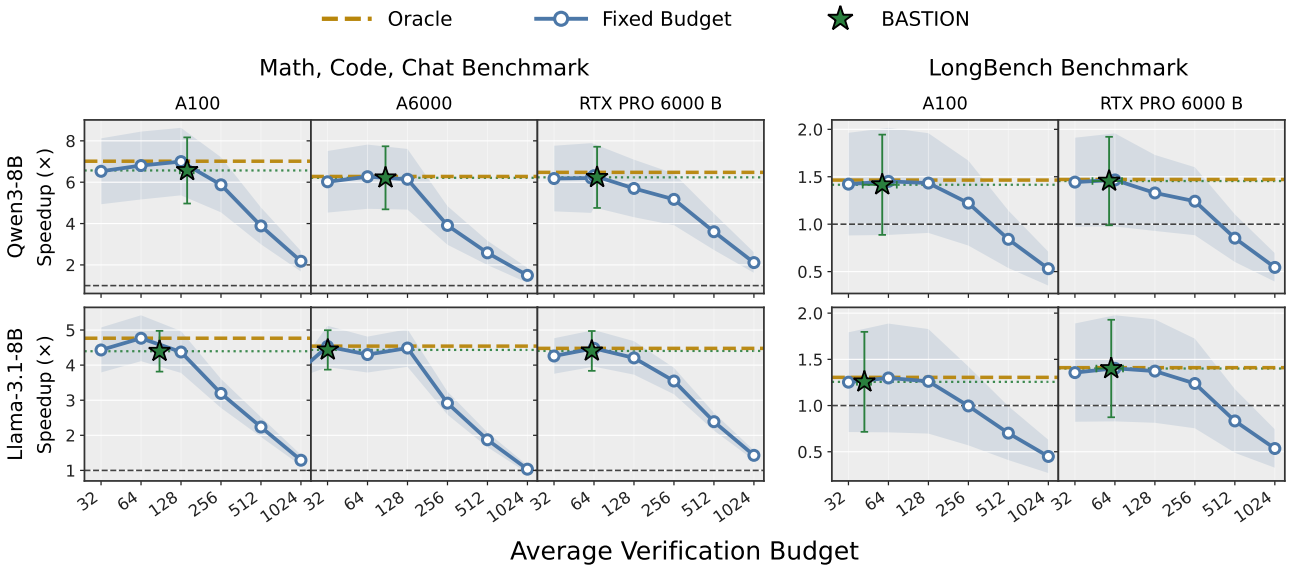


Figure 4. **Budget-policy sweep within BASTION.** Mean speedup over AR decoding at $T{=}0$. **Blue:** BASTION-Fixed ($N{\in}\{32, 64, 128, 256, 512, 1024\}$). **Green stars:** BASTION (mean realized budget). **Dashed gold (*Oracle*):** best per-dataset fixed N averaged per panel—an upper bound for static N without tuning. *Left:* short-context benchmarks over $\{$A100,B6000,A6000$\}\times\{$Qwen3-8B, Llama-3.1-8B-Instruct$\}$; *Right:* LongBench (English) over $\{$B6000 ,A100$\}\times$ same targets. BASTION tracks the Oracle on most panels, recovering near-best speedup without tuning, while over-budgeting degrades the fixed sweep.

Average acceptance length. Across benchmarks, BASTION obtains substantially larger AAL than DFlash, indicating that tree-structured verification accepts longer continuations than single-path drafting. This confirms that the acceptance surrogate and best-first expansion allocate verification nodes to useful candidate paths. The speedup gain is not determined by AAL alone, however: the selected budget must also avoid excessive verifier latency, which motivates the budget-sweep analysis in Section 3.3.

3.3. Fixed vs. Adaptive Budget Policies

Fixed-budget tree construction exposes strong budget sensitivity. We first sweep the verification budget N for the fixed-budget variant (best-first expansion with manually chosen N; denoted *Fixed* in Figure 4). The resulting speedup curves are sharply peaked: small N underutilizes the verifier, large N inflates verification latency faster than it improves accepted length, and the peak location varies across models, GPUs, and benchmarks, proving that a fixed budget is a workload- and hardware-dependent system decision.

BASTION selects near-efficient budgets without per-setting tuning. BASTION replaces the manual choice with an online estimate of the acceptance–latency trade-off. Across all panels in Figure 4, BASTION tracks the per-setting Oracle—the upper bound any fixed N could achieve in hindsight—eliminating the need for a per-setting sweep by construction.

4. Limitations

There are two limitations in our work:

- **Batch size constraints:** Our evaluation targets batch size 1; serving regimes with larger batches alter the verification-cost profile and warrant dedicated treatment.
- **Runtime profile calibration:** Our default offline calibration assumes a stable runtime profile; while BASTION with EMA from static calibration provides a residual-correction fallback, deployments with pronounced runtime drift remain an open direction.

The broader framework—surrogate-driven tree construction with online budget control—extends naturally to other multi-token prediction paradigms beyond block diffusion.

5. Conclusion

We presented BASTION, a budget-aware tree-based speculative decoding method for block-diffusion drafters. BASTION combines a path-confidence surrogate with provably optimal best-first tree construction and a hardware-calibrated online budget controller. Across three targets, four GPUs, and diverse benchmarks, BASTION achieves up to $6.61\times$ speedup over autoregressive decoding ($1.39\times$ over DFlash, $2.45\times$ over EAGLE-3) without per-deployment tuning.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. 1

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. Pard: Accelerating llm inference with low-cost parallel draft model adaptation. *arXiv preprint arXiv:2504.18583*, 2025. 11, 12

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025. 1

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. 20

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025. 1, 11

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5910–5924, 2023. 1, 11

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. URL https://arxiv.org/abs/2308.14508. 20

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024. 1, 2, 11, 12

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. 1, 11, 12

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026. 1, 2, 11, 12, 20, 22

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. 20

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, Robust, and Hardware-aware Speculative Decoding, July 2025. URL http://arxiv.org/abs/2402.12374. arXiv:2402.12374 [cs]. 11, 12

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168. 20

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 1

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.n

aacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/. 20

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. 1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019. 11

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, 2024. 1, 11

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1102. URL https://aclanthology.org/P19-1102/. 20

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024. 11

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In Lu Wang, Jackie Chi Kit Cheung, Giuseppe Carenini, and Fei Liu, editors, *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5409. URL https://aclanthology.org/D19-5409/. 20

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024. 1

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1, 20

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017. 2

Yue Guan, Changming Yu, Shihan Fang, Weiming Hu, Zaifeng Pan, Zheng Wang, Zihan Liu, Yangjie Zhou, Yufei Ding, Minyi Guo, and Jingwen Leng. Yggdrasil: Bridging dynamic speculation and static runtime for latency-optimal tree-based llm decoding, 2025. URL https://arxiv.org/abs/2512.23858. 12

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11575–11596, 2023. 2

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022. 11

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 20

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024. 12

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112/. 20

Hunyuan AI Infra Team. Angelslim: A more accessible, comprehensive, and efficient toolkit for large model compression. *arXiv preprint arXiv:2602.21233*, 2026. 20

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama,

Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. 20

Doohyuk Jang, Sihwan Park, June Yong Yang, Yeonsung Jung, Jihun Yun, Souvik Kundu, Sung-Yub Kim, and Eunho Yang. Lantern: Accelerating visual autoregressive models with relaxed speculative decoding. *arXiv preprint arXiv:2410.03355*, 2024. 1

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/. 20

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36: 39236–39256, 2023. 1, 11, 12

John Kirchenbauer, Abhimanyu Hans, Brian Bartoldson, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Multi-token prediction via self-distillation. *arXiv preprint arXiv:2602.06019*, 2026. 1

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023. 1, 11, 12

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding. *arXiv preprint arXiv:2510.02358*, 2025a. 11, 12

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35:4328–4343, 2022. 2

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a. 1, 2, 11, 12

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024b. URL https://arxiv.org/abs/2406.16858. 2, 12

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025b. 1, 2, 11, 12, 20

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025a. 1, 11, 12

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. *arXiv preprint arXiv:2408.11850*, 2024. 12

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, Xiao Sun, and Xiaoyan Sun. Logitspec: Accelerating retrieval-based speculative decoding via next next token speculation. *arXiv preprint arXiv:2507.01449*, 2025b. 12

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024. 1, 2, 11, 12

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025. 1

Sihwan Park, Doohyuk Jang, Sungyub Kim, Souvik Kundu, and Eunho Yang. Lantern++: Enhancing relaxed speculative decoding with static tree drafting for visual autoregressive models. *arXiv preprint arXiv:2502.06352*, 2025. 2

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026. 12

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. *arXiv preprint arXiv:2408.11049*, 2024. 1, 11

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025. 1, 11, 12

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/. 11

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36:30222–30242, 2023. 12

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 20

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025. 12

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025a. 1

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025b. 1

Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, Jian Jiao, Juntao Li, Jian Guo, Nan Duan, Weizhu Chen, et al. Ar-diffusion: Auto-regressive diffusion model for text generation. *Advances in Neural Information Processing Systems*, 36:39957–39974, 2023. 2

Yangchao Wu, Zongyue Qin, Alex Wong, and Stefano Soatto. Stree: Speculative tree decoding for hybrid state-space models. *arXiv preprint arXiv:2505.14969*, 2025c. 12

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 1, 20

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, 2024a. 1

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. *arXiv preprint arXiv:2412.18910*, 2024b. 12

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025. 20

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023. 20

A. Preliminary

A.1. Speculative Decoding

Mechanism overview. Speculative decoding accelerates generation by decoupling proposal generation from target-model verification (Chen et al., 2023; Leviathan et al., 2023; Cai et al., 2024; Li et al., 2024a). Instead of invoking the expensive target model sequentially, a lightweight draft model first proposes a sequence of future tokens. The target model then verifies these candidates in parallel with a single forward pass. Standard speculative sampling (Chen et al., 2023; Leviathan et al., 2023) probabilistically accepts tokens to theoretically match the target distribution. However, many recent approaches simplify this step by adopting exact match verification (Chen et al., 2026) or relaxed acceptance rules (Cai et al., 2024; Bachmann et al., 2025), bypassing the complex rejection sampling overhead while maintaining high throughput.

Algorithm variants. Subsequent variants differ in their choice of drafters—ranging from model-free (Saxena, 2023; Fu et al., 2024) and autoregressive (Leviathan et al., 2023; Bae et al., 2023; Elhoushi et al., 2024; Li et al., 2025b) to diffusion-based approaches (Samragh et al., 2025; Liu et al., 2025a; Chen et al., 2026). While autoregressive drafters generate candidates through strict left-to-right sequential prediction, diffusion-based drafters instead produce draft spans by jointly denoising multiple future positions in parallel. Prior works have also explored various draft structures, such as sequential chains (Chen et al., 2023; Kim et al., 2023; Sadhukhan et al., 2024) and tree-based topologies (Miao et al., 2024; Cai et al., 2024; Chen et al., 2025). Whereas sequential chains propose a single linear trajectory, tree-based methods simultaneously explore multiple diverging paths to increase the overall acceptance probability. See Appendix B for more details on related work.

Efficiency analysis. Speculative decoding mitigates the memory-bandwidth bottleneck in LLM inference by using idle compute resources to verify multiple draft tokens in a single pass, trading extra computation for fewer memory accesses (Chen et al., 2023; Bae et al., 2023; Sadhukhan et al., 2024). Its practical speedup relative to standard decoding is defined as $\text{Speedup} = \bar{A}T_{\text{AR}}/(T_{\text{draft}} + T_{\text{verify}})$. This demonstrates that efficiency is maximized when the average number of accepted tokens per iteration ($\bar{A}$) is high, and the overhead times for generating drafts (T_{draft}) and verifying them (T_{verify}) are kept minimal compared to the latency of a standard single-token autoregressive step (T_{AR}).

A.2. Block Diffusion Drafter

Drafting algorithm. Unlike autoregressive drafting, which factorizes draft generation into sequential conditionals, one-step block diffusion drafting yields position-wise marginals under a shared masked context (Liu et al., 2025a; Li et al., 2025a; An et al., 2025; Chen et al., 2026). A lightweight drafter—typically 5 to 8 layers—appends a sequence of `[MASK]` tokens to the context and denoises them simultaneously in a single forward pass, equivalent to masked token prediction (Devlin et al., 2019; He et al., 2022). Recent methods enhance drafter accuracy by injecting hidden representations from across the target model's layers directly into the drafter's key-value caches (Liu et al., 2025a; Chen et al., 2026). To extract a final draft trajectory from these parallel predictions, prior works employ greedy decoding, independently taking the top-1 prediction (i.e., `argmax`) at each masked position (Liu et al., 2025a; Chen et al., 2026).

Efficiency analysis. Recall that speculative speedup heavily depends on minimizing the drafting overhead T_{draft}. For an autoregressive drafter, generating a draft block of γ candidates requires γ sequential forward passes, yielding a latency of $T_{\text{draft}} = \gamma \cdot t_{\text{step}}$, where t_{step} is the latency of a single forward pass. A block diffusion drafter circumvents this sequential bottleneck by processing the entire block in a single pass. Since LLM decoding is predominantly memory-bound, evaluating multiple tokens simultaneously incurs minimal computational overhead, effectively reducing the drafting latency to a near-constant $T_{\text{draft}} \approx t_{\text{step}}$.

B. Related Works

Block diffusion drafter for speculative decoding. Recent speculative decoding methods have transitioned from autoregressive drafting (Chen et al., 2023; Leviathan et al., 2023; Kim et al., 2023; Li et al., 2024a) toward multi-token prediction and block diffusion paradigms (Cai et al., 2024; Samragh et al., 2025; Li et al., 2025a; Liu et al., 2025a; Chen et al., 2026). By leveraging the inherent multi-token predictive capacity of modern LLMs (Samragh et al., 2025), parallel drafting maximizes hardware utilization and reduces latency via single forward-pass generation. For example, PARD (An et al., 2025) converts autoregressive drafters into target-independent parallel models via a conditional drop-token mechanism, while DiffuSpec (Li et al., 2025a) enables training-free integration of pretrained diffusion models using causal-consistency path search. Other works directly pretrain block diffusion drafters: TiDAR (Liu et al., 2025a) synergizes diffusion-style drafting with causal verification, and DFlash (Chen et al., 2026) leverages the target model's context features (Li et al., 2025b) via KV injection.

Tree-based speculative decoding. Tree-based speculative decoding improves verifier utilization by replacing a single draft trajectory with a prefix tree, where root-to-leaf paths represent alternative continuations. SpecInfer (Miao et al., 2024) pioneers the verification of such trees through tree attention, while methods like Medusa (Cai et al., 2024), EAGLE (Li et al., 2024a), and EAGLE-2 (Li et al., 2024b) construct multi-token or tree-structured drafts using auxiliary heads or feature-level autoregressive models. To maximize expected acceptance length under hardware constraints, approaches like Sequoia (Chen et al., 2025) and SpecTr (Sun et al., 2023) optimize fixed tree topologies offline through exact search or dynamic programming algorithms using profiling statistics. This static-tree paradigm has also been extended to hybrid state-space models (Wu et al., 2025c).

Adaptive draft tree structure. Recent methods further optimize acceptance efficiency by dynamically adjusting draft tree topologies (Huang et al., 2024; Liu et al., 2024; Wang et al., 2025; Guan et al., 2025; Liu et al., 2025b). While Sequoia (Chen et al., 2025) relies on offline profiling, subsequent approaches introduce online adaptation by modifying draft structures via expectation maximization (OPT-Tree (Wang et al., 2025)), Markov decision processes (SpecDec++ (Huang et al., 2024)), parallel serving (PEARL (Liu et al., 2024)), length predictors (AdaEAGLE (Zhang et al., 2024b)), or latency-aware optimization (Yggdrasil (Guan et al., 2025)). Concurrent with our work, DDTree (Ringel and Romano, 2026) also extends this adaptability to block diffusion drafters via best-first tree expansion. Our work shares a similar tree-based block diffusion drafting paradigm; however, whereas DDTree relies on fixed budgets and manual tuning, we further introduce a hardware-aware adaptive controller to dynamically align the draft structure with real-time system constraints.

C. Mathematical Proofs

C.1. Proofs of Lemma 2.1 and Proposition 2.2

We adopt the notation of §2.2. Let $\mathcal{T}$ be a prefix-closed tree with root r, and let $X = (X_1, \ldots, X_\gamma)$ denote a sample drawn from the drafter's joint distribution; by property (P2), $X_k \sim q_k(\cdot)$ independently across positions. For a node $i \in \mathcal{T}$ at depth $d(i)$ with root-to-node path $\mathrm{path}(i) = (x_{i_1}, \ldots, x_{i_{d(i)}})$, define the *cover indicator*

$$\mathrm{cov}_i(X) \;:=\; \mathbf{1}\big\{X_{1:d(i)} = \mathrm{path}(i)\big\}\,,$$

with the convention $\mathrm{cov}_r(X) = 1$ (the empty prefix is vacuously matched). Let $\mathcal{C}(X) := \{i \in \mathcal{T} : \mathrm{cov}_i(X) = 1\}$ denote the covered set. Under the self-verification model, a node is committed iff it is covered, so the committed length is $A_{\mathrm{self}}(\mathcal{T}; X) = |\mathcal{C}(X)|$.

PROOF OF LEMMA 2.1

We establish three structural properties of $\mathcal{C}(X)$, from which the lemma follows.

(i) Ancestor-closure. Let $i \in \mathcal{C}(X)$ and let j be an ancestor of i in $\mathcal{T}$. Since j lies on the root-to-i path, $\mathrm{path}(j)$ is a prefix of $\mathrm{path}(i)$, so $\mathrm{path}(i)_{1:d(j)} = \mathrm{path}(j)$. Therefore

$$X_{1:d(j)} \;=\; \mathrm{path}(i)_{1:d(j)} \;=\; \mathrm{path}(j),$$

which gives $j \in \mathcal{C}(X)$.

(ii) At most one covered node per depth. Suppose two distinct nodes $i_1, i_2 \in \mathcal{T}$ at depth k both lie in $\mathcal{C}(X)$. Then $X_{1:k} = \mathrm{path}(i_1)$ and $X_{1:k} = \mathrm{path}(i_2)$, forcing $\mathrm{path}(i_1) = \mathrm{path}(i_2)$. But in a tree, two distinct nodes at the same depth necessarily have distinct root-to-node paths: tracing from the root, their paths must agree up to their nearest common ancestor and then diverge into different children. Contradiction.

(iii) Chain structure. Set $m := \max_{i \in \mathcal{C}(X)} d(i)$; the maximum is well-defined because $r \in \mathcal{C}(X)$. By (ii), there exists a unique node $i^\star \in \mathcal{C}(X)$ with $d(i^\star) = m$. By (i), every ancestor of $i^\star$ in $\mathcal{T}$ also belongs to $\mathcal{C}(X)$, contributing exactly one covered node at each depth $k \in \{0, 1, \ldots, m\}$. By (ii), no node outside the root-to-$i^\star$ chain can be in $\mathcal{C}(X)$. Hence $\mathcal{C}(X)$ is exactly the chain from r to $i^\star$, and $|\mathcal{C}(X)| = m + 1$.

Combining the three properties,

$$A_{\mathrm{self}}(\mathcal{T}; X) \;=\; |\mathcal{C}(X)| \;=\; \#\big\{i \in \mathcal{T} : X_{1:d(i)} = \mathrm{path}(i)\big\}\,. \quad \square$$

PROOF OF PROPOSITION 2.2

By Lemma 2.1, the committed length decomposes as a sum of cover indicators:

$$A_{\mathrm{self}}(\mathcal{T}; X) \;=\; \sum_{i \in \mathcal{T}} \mathrm{cov}_i(X).$$

Taking expectations over X and applying linearity,

$$\mathbb{E}_X[A_{\mathrm{self}}(\mathcal{T}; X)] \;=\; \sum_{i \in \mathcal{T}} \mathbb{E}_X[\mathrm{cov}_i(X)] \;=\; \sum_{i \in \mathcal{T}} \mathbb{P}\big(X_{1:d(i)} = \mathrm{path}(i)\big)\,.$$

For the root, $\mathbb{P}(X_{1:0} = \mathrm{path}(r)) = 1 = \rho(r)$ by convention. For any non-root i with $\mathrm{path}(i) = (x_{i_1}, \ldots, x_{i_{d(i)}})$, position-wise independence (P2) yields

$$\mathbb{P}\big(X_{1:d(i)} = \mathrm{path}(i)\big) \;=\; \prod_{k=1}^{d(i)} \mathbb{P}(X_k = x_{i_k}) \;=\; \prod_{k=1}^{d(i)} q_k(x_{i_k}) \;=\; \rho(i).$$

Summing over $\mathcal{T}$,

$$\widehat{A}(\mathcal{T}) \;:=\; \mathbb{E}_X[A_{\mathrm{self}}(\mathcal{T}; X)] \;=\; \sum_{i \in \mathcal{T}} \rho(i),$$

which establishes (3). $\square$

C.2. Proof of Proposition 2.3

Let $i_1, i_2, \ldots$ be the nodes added by best-first expansion in order, with $i_1 = r$.

Step 1: Path monotonicity. For any non-root i with parent $\pi(i)$,

$$\rho(i) \;=\; \rho(\pi(i)) \cdot q_{d(i)}(x_i) \;\le\; \rho(\pi(i)),$$

since $q_{d(i)}(x_i) \in [0, 1]$.

Step 2: Best-first selects in globally non-increasing order of ρ. Suppose, for contradiction, that there exist indices $m < n$ with $\rho(i_m) < \rho(i_n)$. Among such pairs, choose the one minimizing m. At step m, best-first selected i_m from the available frontier F_m (nodes whose parent has been added but which are not yet in $\mathcal{T}_{m-1}$). Consider the root-to-i_n path in the lattice $\mathcal{V}$, and let w be the *first* node on this path that is not in $\{i_1, \ldots, i_{m-1}\}$. Then $\pi(w) \in \{i_1, \ldots, i_{m-1}\}$, so $w \in F_m$. By Step 1 applied along the path from w down to i_n,

$$\rho(w) \;\ge\; \rho(i_n) \;>\; \rho(i_m).$$

But best-first chose i_m as the available node of maximal ρ, contradicting $\rho(w) > \rho(i_m)$ with $w \in F_m$. Hence $\rho(i_1) \ge \rho(i_2) \ge \cdots$.

Step 3: Optimality via leaf exchange. Intuitively, Step 2 already implies that $\mathcal{T}_N$ collects the N largest path scores reachable under prefix-closure; we now make this rigorous by showing that any prefix-closed tree of size N can be transformed into $\mathcal{T}_N$ via a sequence of score-non-decreasing leaf-for-leaf exchanges.

Define $\mathcal{T}_N := \{i_1, \ldots, i_N\}$. Note $\mathcal{T}_N$ is prefix-closed: for any i_k with $k \le N$, best-first added $\pi(i_k)$ before i_k, so $\pi(i_k) \in \mathcal{T}_N$.

Let $\mathcal{T} \subseteq \mathcal{V}$ be any prefix-closed tree with $|\mathcal{T}| = N$ and $\mathcal{T} \neq \mathcal{T}_N$. We construct a prefix-closed $\mathcal{T}'$ with $|\mathcal{T}'| = N$, $\widehat{A}(\mathcal{T}') \ge \widehat{A}(\mathcal{T})$, and $|\mathcal{T}' \triangle \mathcal{T}_N| < |\mathcal{T} \triangle \mathcal{T}_N|$.

Selecting the exchange pair. Let $y \in \mathcal{T} \setminus \mathcal{T}_N$ be a node of maximum depth in $\mathcal{T} \setminus \mathcal{T}_N$. Then y has no children in $\mathcal{T}$: any child c of y in $\mathcal{T}$ has $d(c) > d(y)$, so by maximality $c \notin \mathcal{T} \setminus \mathcal{T}_N$, i.e., $c \in \mathcal{T}_N$; but prefix-closure of $\mathcal{T}_N$ then forces $y = \pi(c) \in \mathcal{T}_N$, contradicting $y \in \mathcal{T} \setminus \mathcal{T}_N$. Hence y is a leaf of $\mathcal{T}$ and $\mathcal{T} \setminus \{y\}$ is prefix-closed.

Let s be the smallest index such that $i_s \notin \mathcal{T}$. Such s exists because $|\mathcal{T}| = |\mathcal{T}_N| = N$ and $\mathcal{T} \neq \mathcal{T}_N$. By minimality, $\{i_1, \ldots, i_{s-1}\} \subseteq \mathcal{T}$, and $\pi(i_s) \in \{i_1, \ldots, i_{s-1}\} \subseteq \mathcal{T}$.

Constructing $\mathcal{T}'$. Define $\mathcal{T}' := (\mathcal{T} \setminus \{y\}) \cup \{i_s\}$. We verify the required properties:

Prefix-closure. (a) Removing y preserves prefix-closure since y is a leaf. (b) Adding i_s requires $\pi(i_s) \in \mathcal{T}'$. We have $\pi(i_s) \in \{i_1, \ldots, i_{s-1}\} \subseteq \mathcal{T} \setminus \{y\}$ (using $\pi(i_s) \in \mathcal{T}_N$ but $y \notin \mathcal{T}_N$, hence $\pi(i_s) \neq y$). So $\pi(i_s) \in \mathcal{T}'$.

Score improvement. Since $i_s \in \mathcal{T}_N$ and $y \notin \mathcal{T}_N$, in best-first order the index of i_s is at most N while the index of y exceeds N. By Step 2, $\rho(i_s) \ge \rho(y)$. Therefore

$$\widehat{A}(\mathcal{T}') \;=\; \widehat{A}(\mathcal{T}) - \rho(y) + \rho(i_s) \;\ge\; \widehat{A}(\mathcal{T}).$$

Symmetric difference decreases by 2. $\mathcal{T}'$ removes y (which was in $\mathcal{T} \setminus \mathcal{T}_N$) and adds i_s (which was in $\mathcal{T}_N \setminus \mathcal{T}$).

Iterating this exchange yields a finite sequence of prefix-closed trees of size N with non-decreasing $\widehat{A}$, terminating at $\mathcal{T}_N$. Therefore $\widehat{A}(\mathcal{T}_N) \ge \widehat{A}(\mathcal{T})$.

Step 4: Diminishing returns. (5) is immediate from Step 2: $\Delta\widehat{A}(N) = \rho(i_{N+1})$, and the global non-increasing order gives $\rho(i_{N+2}) \le \rho(i_{N+1})$. □

C.3. Proof of Proposition 2.4

Let $S(N) = L_{\mathrm{AR}} \cdot \widehat{A}(N)/C(N)$ on $\{1, \ldots, N_{\max}\}$. Recall the assumptions:

(A1) $\widehat{A}(N)$ is non-decreasing and concave (Proposition 2.3, (5)).

(A2) $C(N) > 0$ is convex and non-decreasing.

Without loss of generality drop the positive constant L_{AR} and study $f(N) = \widehat{A}(N)/C(N)$.

Step 1: Continuous extension. Linearly interpolate $\widehat{A}$ and C to functions on $[1, N_{\max}]$. The interpolated $\widehat{A}$ remains concave and non-negative; the interpolated C remains convex and strictly positive. Both extensions are continuous.

Step 2: Quasiconcavity of f. A function is quasiconcave iff every upper level set is convex. For any $c \geq 0$,

$$f(x) \geq c \iff \widehat{A}(x) - c\,C(x) \geq 0.$$

The function $g_c(x) := \widehat{A}(x) - c\,C(x)$ is concave: $\widehat{A}$ is concave by (A1), $-c\,C$ is concave because $c \geq 0$ and C is convex by (A2), and the sum of concave functions is concave. The upper level set $\{x : g_c(x) \geq 0\}$ of a concave function is convex. Hence $\{x : f(x) \geq c\}$ is convex for every c, so f is quasiconcave on $[1, N_{\max}]$.

Step 3: Continuous unimodality. A continuous quasiconcave function on a real interval is unimodal: there exists $x^\star \in [1, N_{\max}]$ such that f is non-decreasing on $[1, x^\star]$ and non-increasing on $[x^\star, N_{\max}]$. Otherwise, one could find $x_1 < x_2 < x_3$ with $f(x_2) < \min(f(x_1), f(x_3))$, and the upper level set at level $\min(f(x_1), f(x_3))$ would contain x_1 and x_3 but not x_2, contradicting convexity.

Step 4: Discrete unimodality. Restrict to integers. Let $N^\star \in \arg\max_{N \in \{1,\ldots,N_{\max}\}} S(N)$. By Step 3, S is non-decreasing on integers in $[1, x^\star]$ and non-increasing on integers in $[x^\star, N_{\max}]$, so S is non-decreasing on $\{1, \ldots, N^\star\}$ and non-increasing on $\{N^\star, \ldots, N_{\max}\}$.

Step 5: First-decrease optimality. Suppose N is the smallest index with $S(N{+}1) < S(N)$. Then S has not yet decreased on $\{1, \ldots, N\}$, so $S(1) \leq S(2) \leq \cdots \leq S(N)$. By Step 4, since S decreases at $N{+}1$, we must have $N \geq N^\star$, and combined with non-decreasing-up-to-N, $N = N^\star$. □

D. Algorithm

Algorithm 1 System-aware Adaptive Tree Construction at Cycle t

Input: Block size γ; drafter probabilities $q_{t,1:\gamma}$; root/bonus node r_t; context size c_t; maximum verification budget $N_{\max}$; draft latency $T_{\text{draft},t}$; auxiliary-latency model $T_{\text{aux},t}(\cdot)$; AR latency L_{AR}.

Output: Selected verification tree $\mathcal{T}_t^\star$ and budget $N_t^\star$.

Initialize $\mathcal{T} \leftarrow \{r_t\}$ and $\widehat{\widehat{A}} \leftarrow 1$ `// root/bonus contribution`
Initialize $\mathcal{T}_t^\star \leftarrow \mathcal{T}$, $N_t^\star \leftarrow 0$, and $\widehat{S}^\star \leftarrow -\infty$
Initialize max-heap H with the highest-probability depth-1 draft node, keyed by path score $\rho(\cdot)$
for $N = 1$ **to** $N_{\max}$ **do**
  if H *is empty* **then**
    break
  Pop the available node u with largest path score $\rho(u)$ from H
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{u\}$ $\widehat{\widehat{A}} \leftarrow \widehat{\widehat{A}} + \rho(u)$
  `/* Evaluate the estimated speedup after adding node` u`. */`
  $\widehat{T}_{\text{verify}} \leftarrow \textsc{EstimateLatency}(N, c_t)$ $\widehat{C} \leftarrow T_{\text{draft},t} + \widehat{T}_{\text{verify}} + T_{\text{aux},t}(N)$
  $\widehat{S} \leftarrow \widehat{\widehat{A}} \cdot L_{\text{AR}} / \widehat{C}$
  if $\widehat{S} > \widehat{S}^\star$ **then**
    $\widehat{S}^\star \leftarrow \widehat{S}$
    $\mathcal{T}_t^\star \leftarrow \mathcal{T}$
    $N_t^\star \leftarrow N$
  else
    break
  `/* Update the frontier of the implicit prefix lattice. */`
  if $\text{depth}(u) < \gamma$ **then**
    Push the highest-probability child of u into H
  if u *has an unpushed next sibling under the same parent* **then**
    Push the next sibling of u into H
return $\mathcal{T}_t^\star, N_t^\star$

E. Overall Pipeline

Figure 5 (bottom) summarizes one iteration of our pipeline; we walk through its four stages below.

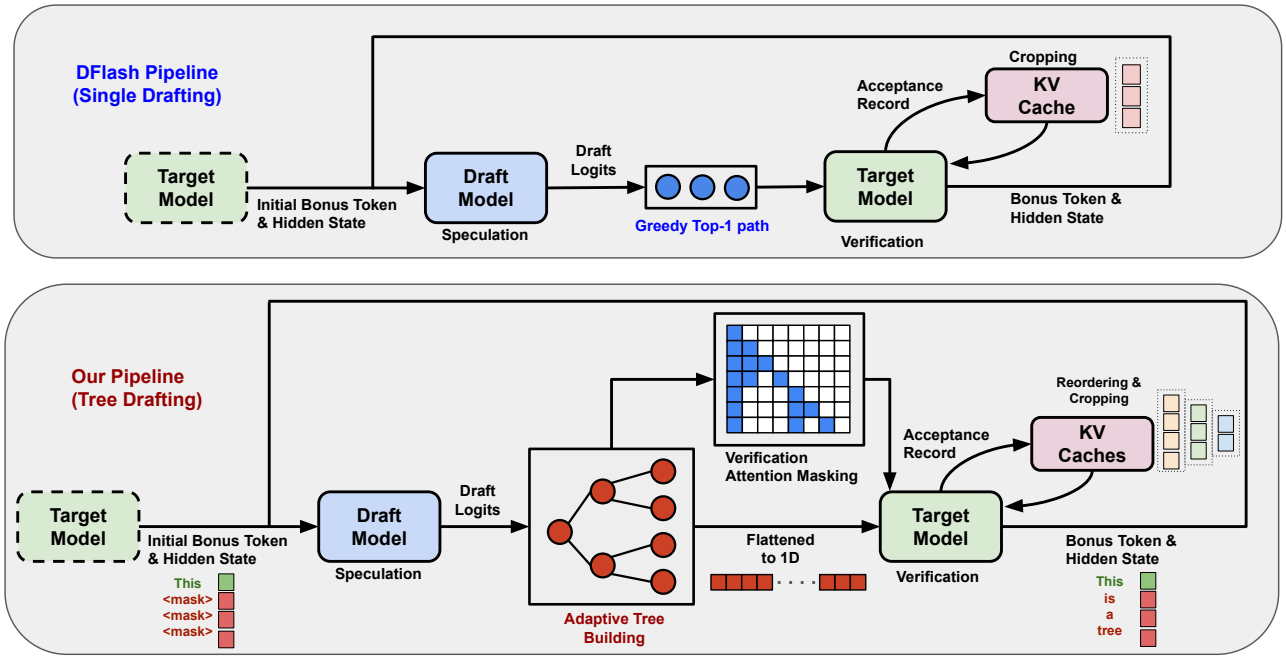


Figure 5. **Single-path vs. tree-based speculative decoding.** *Top:* the DFlash baseline drafts a single greedy top-1 continuation, which the target verifies in one forward pass before cropping the KV cache to the accepted prefix. *Bottom:* our tree pipeline. From the same initial bonus token and hidden state, the draft model emits per-position logits that the adaptive tree builder expands into a token tree. The tree is linearized to a 1D input for the target, and a custom verification attention mask preserves parent–child dependencies in a single batched forward pass. The acceptance record selects the best path through the tree; the target's KV cache is then reordered and cropped to retain only accepted nodes, and the new bonus token and hidden state feed the next iteration.

Drafting. Each decoding step begins from the bonus token and hidden state produced by the previous target forward pass, or, on the first step, from a short prefill. The draft model consumes these and emits a distribution over candidate continuations at multiple future positions. Rather than committing to a single greedy path as in the baseline (Figure 5, top), we feed these draft distributions into an adaptive tree builder that grows a token tree under a fixed verification budget, allocating depth and width to the most promising branches.

Linearization and verification mask. The tree is linearized into a flat sequence for the target model. To make a single batched forward pass behave as if each branch were verified independently, we construct a custom attention mask (Figure 5, center-bottom): every tree node attends to the shared prefix in the KV cache and to its own ancestors in the tree, but not to siblings or unrelated branches. Position ids are derived from each node's depth, so rotary embeddings see the correct relative position regardless of the linearization order.

Acceptance. After the target forward pass, we score every root-to-leaf path against the target's greedy top-1 predictions and select the longest prefix that the target itself would have produced. The accepted nodes' tokens are emitted, and the target's KV cache is reordered and cropped so that it contains the shared prefix followed only by the accepted path—discarding the rest of the tree.

Iteration. The hidden state of the deepest accepted node, together with the target's next-token sample at that position, becomes the bonus token and hidden state that seed the next iteration. The cycle repeats until an EOS token or length limit is reached.

F. Analytical Cost Modeling

We estimate verifier latency with a roofline-style predictor,

$$R_t(N) = \max\left(\frac{\text{FLOPs}(N, C_t)}{\text{PeakFLOPs}}, \frac{\text{Bytes}(N, C_t)}{\text{Bandwidth}}\right),$$

so this appendix derives the corresponding FLOP count and memory traffic for a standard Transformer verifier. We assume a non-fused SDPA implementation, count one multiply-accumulate as 2 FLOPs, and omit elementwise operations (e.g., RMSNorm, RoPE, Softmax, SwiGLU nonlinearity), whose cost is lower-order than the dominant matrix multiplications.

Notation. We use the notation in Table 2. Define $h_q = n_q d$ and $h_{kv} = n_{kv} d$.

Table 2. Notation used in the cost model.

Symbol	Meaning
s	number of newly verified tokens
c	cached context length
L	number of transformer layers
h	hidden size
n_q	number of query heads
n_{kv}	number of key/value heads
d	head dimension
h_{ffn}	FFN intermediate size
V	vocabulary size
bp	bytes per parameter / activation element

F.1. FLOPs

Table 3 lists the dominant FLOP terms. Summing them gives the total verifier FLOPs:

$$\text{FLOPs} = L(4shh_q + 4shh_{kv} + 4s(c+s)h_q + 6shh_{ffn}) + 2shV. \tag{8}$$

Table 3. FLOP breakdown for a standard Transformer verifier.

Module	Operation	FLOPs
Attention / layer	Q projection	$2shh_q$
	K, V projections	$4shh_{kv}$
	attention scores $QK^\top$	$2s(c+s)h_q$
	attention-value product	$2s(c+s)h_q$
	output projection	$2sh_qh$
Attention total		$L(4shh_q + 4shh_{kv} + 4s(c+s)h_q)$
FFN / layer	gate/up projections	$4shh_{ffn}$
	down projection	$2sh_{ffn}h$
FFN total		$L(6shh_{ffn})$
LM head	hidden $\rightarrow$ vocab	$2shV$

F.2. Memory Traffic

We decompose memory traffic into (i) model weights, (ii) KV-cache reads/writes, and (iii) intermediate activations. Table 4 summarizes the terms. The resulting total memory traffic is

$$\begin{aligned}\text{Bytes} = bp\Big[&2Vh + s(h+V) \\ &+ L\big(2h(h_q + h_{kv}) + 3hh_{ffn} + 2h_{kv}(c+2s) \\ &\quad + 4s(h + h_q + h_{ffn}) + 2n_q s(c+s)\big)\Big].\end{aligned} \tag{9}$$

Table 4. Memory traffic breakdown for the verifier.

Category	**Module**	**Read / Write**
Weights	Embedding + LM head Attention weights / layer FFN weights / layer	$Vh + hV$ $2hh_q + 2hh_{kv}$ $3hh_{ffn}$
Weights total in Bytes		$bp[L(2hh_q + 2hh_{kv} + 3hh_{ffn}) + 2Vh]$
KV cache	past-context read / layer new-token write / layer	$2ch_{kv}$ $2sh_{kv}$
KV cache total in Bytes		$bp[L(2ch_{kv} + 2sh_{kv})]$
Activations	attention I/O / layer FFN I/O / layer LM head I/O	$2sh + 4sh_q + 2sh_{kv} + 2n_q s(c+s)$ $2sh + 4sh_{ffn}$ $sh + sV$
Activations total in Bytes		$bp[L(4sh + 4sh_q + 2sh_{kv} + 4sh_{ffn} + 2n_q s(c+s))$ $+ sh + sV]$

Remarks. Even when input embeddings and the LM head are tied, they are counted separately in memory traffic because they are accessed at different stages of the forward pass. The formulas above are used only as a lightweight latency predictor; calibration in the main method compensates for implementation-dependent effects such as kernel fusion and cache behavior.

G. Detailed Experimental Setup

Models and datasets. We evaluate BASTION with three autoregressive target models: Qwen3-4B, Qwen3-8B, and Llama-3.1-8B-Instruct (Yang et al., 2025; Grattafiori et al., 2024). Each target model is paired with its compatible block-diffusion drafter from DFlash (Chen et al., 2026). Our evaluation covers four domains. For mathematical reasoning, we use GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), and AIME25 (Zhang and Math-AI, 2025). For code generation, we use HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024). For general instruction following, we use Alpaca (Taori et al., 2023) and MT-Bench (Zheng et al., 2023). For long-context understanding, we use seven English subsets of LongBench (Bai et al., 2024): Qasper (Dasigi et al., 2021), MultiFieldQA-en, GovReport (Huang et al., 2021), MultiNews (Fabbri et al., 2019), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019), and PassageRetrieval-en. Unless stated otherwise, all experiments are run with batch size 1.

Metrics and hardware. Our primary metric is speedup over target-only autoregressive decoding, computed as the ratio between target-only per-token latency and speculative-decoding per-token latency. We also report average accepted length AAL (τ). We evaluate hardware adaptability on four NVIDIA GPUs: A100 (80 GB HBM2e), H100 (80 GB HBM3), RTX PRO 6000 Blackwell (96 GB GDDR7), and RTX A6000 (48 GB GDDR6). All runs use a single GPU at batch size 1; the per-step controller overhead of BASTION is negligible relative to the target forward pass on every tested machine.

Baselines. We compare against DFlash, the most directly related block-diffusion speculative decoding baseline, and EAGLE-3 (Li et al., 2025b), a strong tree-based speculative decoding baseline with an AR-style drafter. For EAGLE-3, we use the open-source AngelSlim (Hunyuan AI Infra Team, 2026) implementation and set the tree size to 60 for all experiments. All baselines use the same target model, benchmark, decoding temperature, and batch size as BASTION.

Compute resources. A single (target, benchmark, method, GPU, temperature) configuration completes in roughly 4 hours on the eight short-context benchmarks and 5 hours on the LongBench English subsets, depending on output length and accepted-tokens-per-step. The full experiment grid—three targets, four GPUs (with not all (model, GPU) cells run; see Tables 1 and 5 and Figures 7, 4), three methods, eight short-context plus seven long-context benchmarks, two temperatures, and the budget sweep ($N \in \{32, 64, 128, 256, 512, 1024\}$ plus BASTION)—requires approximately 12 GPU-hours to reproduce the reported numbers. Development and ablation runs not included in the paper (drafter compatibility checks, latency-model calibration on configurations not reported, alternative scoring functions, and rerun cycles after pipeline fixes) contributed an additional $\sim$1.5–2$\times$ of that compute. No model training was performed: BASTION is training-free, and all target and drafter checkpoints are used as released.

H. Analysis

H.1. Path Score Validation

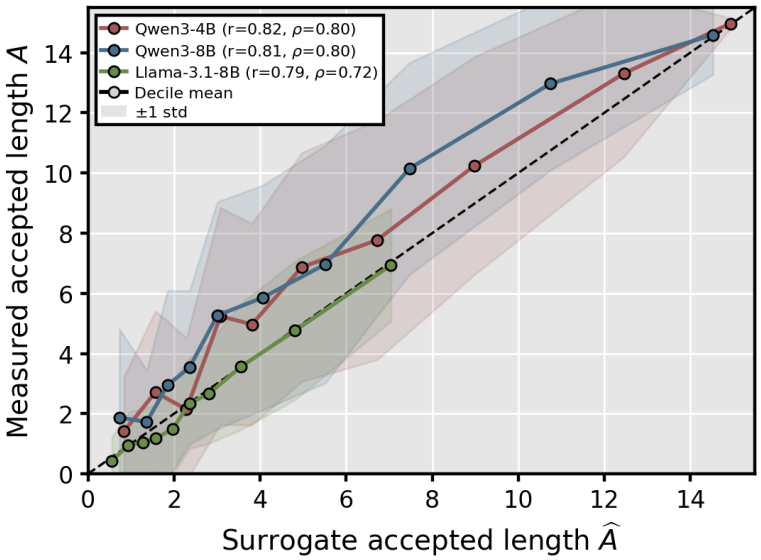


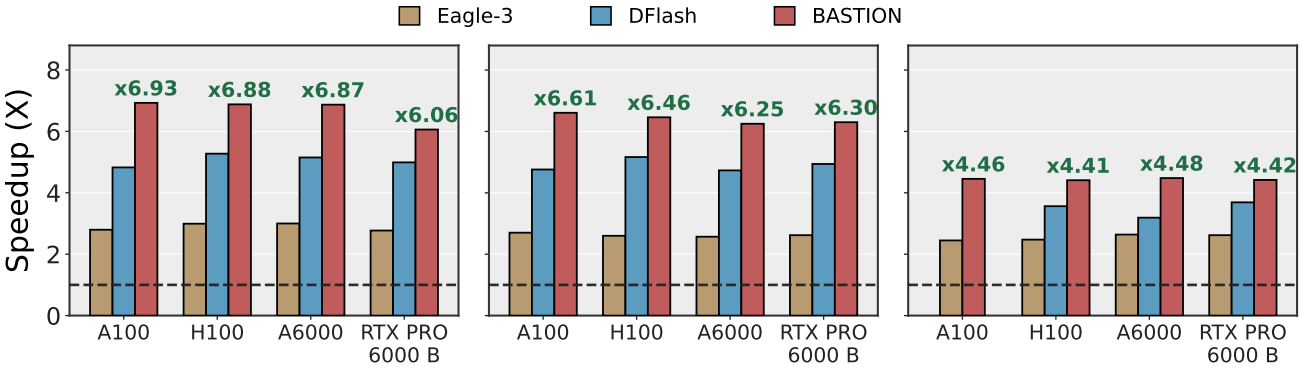
Figure 6. **Path score validation.** For each decode step we collapse the draft tree to a single greedy path by taking the drafter's top-1 token $x_k = \arg\max_v q_k(v)$ at every position $k \in \{1, \ldots, \gamma\}$, where γ is the block size and $q_k(v)$ is the drafter distribution of vocabulary v for position k. Then we evaluate the surrogate accepted length specialized to this path, $\widehat{A} = \sum_{k=1}^{\gamma} \prod_{j=1}^{k} q_j(x_j)$, which is the tree sum of path scores $\rho(i)$ from (1) restricted to a depth-γ chain (root term omitted to match the measured length below). Setup: DFlash block-diffusion drafters paired with each target, evaluated on MATH500 with 20 prompts at temperature 0; block size is 16 for Qwen and 10 for Llama. $\widehat{A}$ correlates strongly with the target-verified accepted length A across all three target/draft pairs (Pearson 0.79–0.82, Spearman 0.72–0.80), supporting the use of drafter path probabilities as a proxy for target acceptance in Section 2.2.

Table 5. **Additional results on H100.** Speedup and AAL (τ) for Qwen3-4B (Q3-4B) and Qwen3-8B (Q3-8B) at temperatures $T = 0$ and $T = 1$. For the baselines, the tree budget size of EAGLE-3 is set to 60. The block size of DFlash is 16 for both Qwen3-4B and Qwen3-8B. **Bold** indicates the highest value within each (model, benchmark) pair.

Model	Method	Math						Code						Chat				Average	
		GSM8K		MATH500		AIME25		HumanEval		MBPP		LCB		MT-Bench		Alpaca			
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Temperature $T = 0$																			
Q3-4B	EAGLE-3	3.23×	2.80	3.04×	2.54	3.13×	2.51	3.29×	2.54	2.95×	2.47	2.86×	2.24	2.89×	2.49	2.55×	2.17	2.99×	2.47
	DFlash	5.83×	6.51	6.96×	7.79	6.45×	7.26	5.87×	6.69	5.40×	6.12	5.98×	6.97	3.21×	4.38	2.49×	3.09	5.27×	6.10
	BASTION	**7.26×**	**8.91**	**8.05×**	**10.20**	**7.96×**	**9.43**	**8.13×**	**9.23**	**7.22×**	**8.81**	**8.37×**	**9.36**	**4.47×**	**6.34**	**3.58×**	**4.73**	**6.88×**	**8.38**
Q3-8B	EAGLE-3	3.16×	2.69	2.83×	2.48	2.85×	2.50	2.43×	2.65	2.82×	2.20	2.17×	1.81	2.49×	2.25	2.05×	1.73	2.60×	2.29
	DFlash	5.87×	6.55	6.86×	7.93	5.95×	7.18	5.83×	6.56	5.27×	5.98	6.04×	7.16	3.03×	4.23	2.47×	3.09	5.17×	6.09
	BASTION	**7.64×**	**9.02**	**8.02×**	**10.21**	**6.88×**	**9.35**	**7.33×**	**9.20**	**7.04×**	**8.62**	**7.30×**	**9.48**	**4.12×**	**6.16**	**3.35×**	**4.82**	**6.46×**	**8.36**
Temperature $T = 1$																			
Q3-4B	EAGLE-3	3.39×	2.75	3.03×	2.41	2.50×	2.16	3.03×	2.41	2.94×	2.38	2.30×	2.07	2.55×	2.38	2.23×	2.08	2.75×	2.33
	DFlash	5.55×	5.98	5.59×	6.60	4.13×	4.97	5.33×	5.93	4.99×	5.63	5.62×	6.57	3.00×	4.01	2.40×	2.95	4.58×	5.33
	BASTION	**8.08×**	**8.78**	**8.50×**	**10.15**	**7.41×**	**9.16**	**8.26×**	**9.19**	**7.68×**	**8.74**	**7.18×**	**8.98**	**4.48×**	**6.21**	**3.59×**	**4.71**	**6.90×**	**8.24**
Q3-8B	EAGLE-3	3.22×	2.61	2.75×	2.35	2.47×	2.06	2.80×	2.53	2.58×	2.21	2.20×	1.79	2.23×	2.10	1.79×	1.62	2.51×	2.16
	DFlash	5.09×	5.82	5.43×	6.45	3.98×	4.83	4.97×	5.58	4.60×	5.26	5.61×	6.82	2.72×	3.84	2.38×	2.94	4.35×	5.19
	BASTION	**7.48×**	**8.98**	**7.59×**	**10.20**	**7.10×**	**8.81**	**7.30×**	**9.15**	**6.66×**	**8.57**	**7.75×**	**9.40**	**4.08×**	**6.18**	**3.39×**	**4.73**	**6.42×**	**8.25**

H.2. Generalization across GPUs and model families

Beyond A100, BASTION retains its lead on the A6000 and RTX PRO 6000 Blackwell across all target models (Qwen3-4B/8B and Llama-3.1-8B-Instruct; Figure 7), as well as on H100 (Table 5). This demonstrates that the improvements are not tied to specific GPU generations or model families.



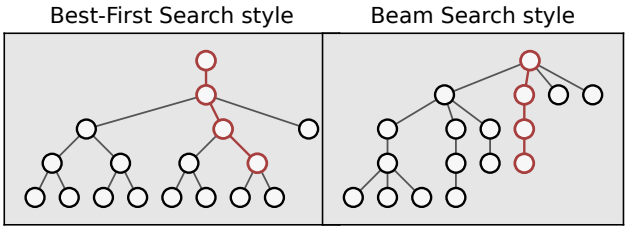
(a) Qwen3-4B *(b)* Qwen3-8B *(c)* Llama-3.1-8B-Instruct

Figure 7. **Additional speedup results across GPU architectures.** Per-cell average wall-clock speedup of BASTION versus EAGLE-3 and DFlash on (a) Qwen3-4B, (b) Qwen3-8B, and (c) Llama-3.1-8B-Instruct, evaluated on four NVIDIA GPUs (A100, H100, A6000, and RTX PRO 6000 Blackwell) at temperature $T = 0$. Each bar reports the mean speedup over autoregressive decoding across all eight benchmarks. Numbers above each red bar give the speedup achieved by BASTION, which dominates both baselines on every (model, GPU) cell.

H.3. Tree Expansion Comparison

Tree topology comparison. We ablate the tree expansion rule under a matched verification budget N, comparing the best-first expansion of BASTION against standard beam search. Unlike beam search, which uniformly retains the top w extensions depth-by-depth to form a rigid $(w \times d)$ topology, best-first dynamically expands the highest-scoring open nodes globally based on cumulative path scores $\rho(i)$ ((1)). Figure 8(a) illustrates this for N=17: beam search yields a fixed rectangle, whereas best-first flexibly deepens high-confidence paths. By Proposition 2.3, best-first produces the optimal prefix-closed tree under the surrogate $\widehat{A}$; this ablation evaluates whether this theoretical optimality translates to wall-clock speedups over beam search. Additionally, Figure 8(b) presents a non-budget-matched single-path DFlash baseline (greedy, block size 16) to isolate the inherent benefits of utilizing a tree structure over a no-tree reference.

Best-first expansion outperforms beam search under matched budgets. Figure 8(b) shows best-first expansion improves mean speedup and average acceptance length τ under a matched budget (beam: w=4, d=15; best-first: N=61). On Qwen3-4B and Qwen3-8B, best-first yields $+7.0\%$ and $+6.1\%$ speedups respectively, with consistently higher τ. While both tree variants outperform the greedy baseline, this comparison isolates topology as the source of best-first's advantage. Concentrating expansion on high-scoring prefixes, rather than uniformly across depths, enables more efficient block-diffusion drafting.



(a) Tree topology under $N = 17$

Model	Method	Tree	Speedup	τ
Q3-4B	greedy (Chen et al., 2026)	✗	5.14×	6.10
	beam search	✓	6.16×	7.77
	best-first	✓	**6.59×**	**8.19**
Q3-8B	greedy (Chen et al., 2026)	✗	4.73×	6.09
	beam search	✓	5.74×	7.60
	best-first	✓	**6.09×**	**7.96**

(b) Average speedup across 8 benchmarks

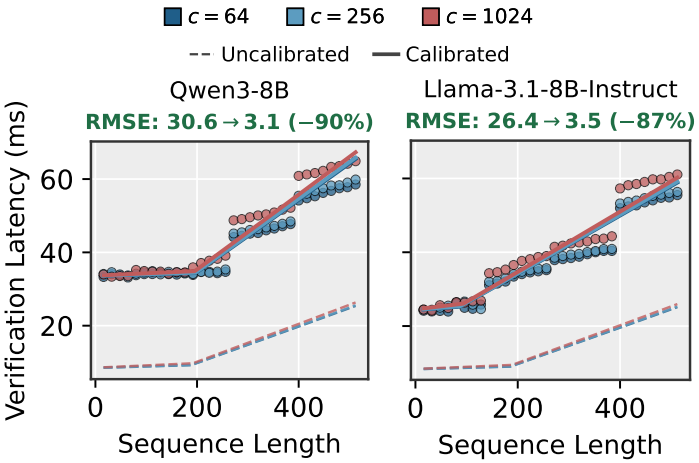
Figure 8. **Tree expansion under fixed budgets. (a)** At N=17, beam search spreads nodes uniformly, while best-first focuses on high-scoring prefixes (red: accepted prefix). **(b)** Average A6000 speedup across 8 math/code/chat benchmarks. Under matched budgets, best-first (N=61) outperforms beam (w=4, d=15), improving Qwen3-4B/8B by $+7.0\%$/$+6.1\%$ (higher τ). Greedy (Chen et al., 2026) (single-path, block 16) is an unmatched no-tree baseline. **Bold** indicates the best per block.

H.4. Latency Model Evaluation

Calibrated roofline as the default latency estimator. Figure 9a shows verification latency as a function of verified sequence length across context lengths $c \in \{64, 256, 1024\}$ for the three targets. The bare roofline (dashed) captures the scaling trend but systematically underestimates absolute latency; the step-function surges reflect CUDA tile-allocation overhead. A lightweight per-(GPU, target) linear calibration (solid) reduces RMSE by 87–92% (29.2→2.3 ms on Qwen3-4B, 30.6→3.1 ms on Qwen3-8B, 26.4→3.5 ms on Llama-3.1-8B). BASTION uses this calibrated curve as its default latency estimator, which we refer to as *Static* below.

Online alternatives when offline calibration is unavailable. For deployments lacking per-(GPU, target) calibration data, we evaluate two online variants. *EMA* discards the offline curve and tracks a single multiplicative bias c via EMA on the ratio of observed to bare-roofline latency, $c \leftarrow (1-\alpha)\, c + \alpha\, (\text{obs}/\text{pred})$, using $c \cdot T_{\text{verify}}(N)$ in place of the raw prediction. *EMA+Calib* applies the same EMA as a residual correction on top of the offline-calibrated curve.

Static is the most robust default; EMA alone is fragile. Figure 9b reports mean speedup, τ, and realized tree size $\bar{N}$ on A100 across eight short-context benchmarks at $T{=}0$. *Static* is best on Llama-3.1-8B and within $0.28\times$ of the best on both Qwen3 targets. *EMA* selects tighter trees on Qwen3 (gaining $+0.28\times$/$+0.18\times$) but settles at *larger* trees on Llama-3.1-8B without an acceptance gain, losing -12.7%. *EMA+Calib* tracks *Static* within $\pm 0.06\times$ everywhere, indicating the offline warm start dominates the EMA residual. We adopt *Static* as the default and *EMA+Calib* as the fallback when offline calibration is unavailable.



(a) Verification-latency calibration

Target	Method	Speedup	τ	$\bar{N}$
Q3-4B	Static	<u>6.93×</u>	<u>8.55</u>	202.0
	EMA+Calib	6.87×	**8.56**	201.9
	EMA	**7.21×**	8.22	102.2
Q3-8B	Static	<u>6.61×</u>	**8.47**	139.3
	EMA+Calib	6.55×	8.44	139.8
	EMA	**6.79×**	<u>8.45</u>	126.6
L3.1-8B	Static	**4.46×**	<u>6.00</u>	87
	EMA+Calib	<u>4.42×</u>	5.99	87
	EMA	3.89×	**6.13**	123.1

(b) Online-calibration variants

Figure 9. **Latency model evaluation on A100. (a)** Verification latency vs. sequence length for three targets at contexts $c \in \{64, 256, 1024\}$. Dashed and solid lines denote the uncalibrated roofline and calibrated fit (used by the controller). Calibration cuts RMSE by 87–92%. **(b)** Mean over 8 short-context benchmarks at $T{=}0$ ($\bar{N}$: mean realized tree size). BASTION variants: *Static* (offline curve), *EMA+Calib* (offline + online EMA residual), and *EMA* (online only). **Bold** and <u>underline</u> indicate the best and second best per model. Highlighted: default BASTION config.